



Europäisches Patentamt
European Patent Office
Office européen des brevets



(11) **EP 1 061 445 A2**

(12) **EUROPEAN PATENT APPLICATION**

(43) Date of publication:
20.12.2000 Bulletin 2000/51

(51) Int. Cl.⁷: **G06F 9/46, H04L 29/06**

(21) Application number: **00304970.7**

(22) Date of filing: **13.06.2000**

(84) Designated Contracting States:
**AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU
MC NL PT SE**
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: **14.06.1999 US 139239 P
14.06.1999 US 333878**

(71) Applicant:
**SUN MICROSYSTEMS, INC.
Palo Alto, California 94303 (US)**

(72) Inventors:
• **Johnson, Thomas V.
Burlington, MA 01803 (US)**
• **Davis, James D.
Pepperell, MA 01463 (US)**

(74) Representative:
**Browne, Robin Forsythe, Dr.
Urquhart-Dykes & Lord
Tower House
Merrion Way
Leeds LS2 8PA West Yorkshire (GB)**

(54) **Web-based enterprise management with transport neutral client interface**

(57) A transport neutral technique allows a management application (32) to communicate with a computer system (110) using any of a variety of network protocols (158, 160). The management application software is independent of the transport mechanism used and need not be changed if the transport mechanism changes. A computer system (110) to be managed includes a CIM object manager (20) and any number of provider APIs (122) that provide resource information about the computer system. A CIM repository (130) stores classes and instances used by the object manager. A remote application computer (150) runs a software management application (32) that communicates with the object manager of the computer system using a local client API (156). The client API of the application computer includes an interface definition (300) defining all methods called by the management application. Also included is a protocol-specific class that implements the interface definition; there is a protocol-specific class for each protocol desired to be supported. Each class implements methods using a specific protocol. A factory class is executed when the management application invokes a method call passing in a desired protocol parameter. The factory class creates a protocol-specific object of one of the protocol-specific classes depending on the protocol parameter. The object is returned to the management application which executes one of its protocol-specific methods thus allowing communication to the computer system using a protocol independent of the management application.

Description

FIELD OF THE INVENTION

[0001] The present invention relates generally to managing the resources of a computer system. More specifically, the present invention relates to a transport neutral technique for communicating requests from a resource management application to a computer system having a Common Information Model (CIM) object manager.

BACKGROUND OF THE INVENTION

[0002] Recently, computers and their associated peripheral equipment (a computer system) have become increasingly more complex. As such, it has become progressively more and more complicated for a user or system administrator to manage the resources of such a computer system. With a variety of peripheral devices and software applications available for use, and their ever-changing nature, the job of a system administrator has become more difficult. Computer system resources such as attached devices, network connections, software applications, etc., must all be managed to ensure an efficiently working system for the user. Within a large corporation having large numbers of such computer systems spread around the world, the task of managing the resources of each computer system can be daunting.

[0003] Recently, the industry has responded to such a need by introducing Web-Based Enterprises Management (WBEM) which is both an initiative and a technology. As an initiative, WBEM includes a standard for managing systems, networks, users, and applications by using Internet technology. As a technology, WBEM provides a way for management applications to share management data independently of vendor, protocol, operating system, or management standard. By developing management applications according to WBEM principles, vendors can develop products that work together easily at a lower cost of development.

[0004] One known standard for implementation of WBEM is the Common Information Model (CIM). CIM is an approach to managing systems and networks. CIM provides a common conceptual framework to classify and define the parts of a network environment and depict how they integrate. The model captures notions that are applicable to all areas of management, independent of technology implementation.

[0005] WBEM software includes tools and technology that software developers can use to create CIM-compliant software applications that manage the environment of a computer system. Developers can also use this software to write "providers," programs that supply data and events for managed objects that are specific to their domain.

[0006] There can be drawbacks, however, associ-

ated with various implementations of WBEM software. Because a management application may be called upon to manage a computer system remotely over a network connection such as the Internet, it would be desirable to perform this task efficiently. Not all implementations, however, are well-suited for various types of network protocols.

[0007] FIG. 1 illustrates a prior art computer system 10 that has resources to be managed. Resources include disk usage, CPU utilization, running applications, etc. Not shown for simplicity are hardware components of the computer system or other software applications. A CIM object manager 20 is responsible for handling all communication between management applications, a CIM repository 26 and managed objects. In this example, CIM repository 26 is a local drive that communicates via a single protocol to object manager 20. A provider application programming interface (API) 22 provides an interface to any needed system information 24 to be provided to object manager 20. Object manager 20 may also access CIM repository 26 over a local interface connection 28 to quickly retrieve data objects that have been previously stored. In this fashion, object manager 20 is well-suited for gathering resource information regarding computer system 10.

[0008] In order to efficiently manage the resources of computer system 10, a software developer 30 writes management application software 32 for managing the resources of the computer system. When in operation, the results of management application 32 may be used by a system administrator 40 to manage the computer system. Client application 32 communicates via a client API 34 to retrieve resource information from computer system 10.

[0009] Prior art implementations of this sort use a single interface protocol 36 for communication with object manager 20. For example, an implementation by Microsoft Corporation implements interface 36 using a Common Object Model (COM) which is inflexible. In particular, client API 34 and COM interface 36 are not completely independent of one another in that there is a one-to-one mapping between client API 34 and COM interface 36. This implies that management application 32 will use COM constructs and language in its implementation. Should the COM interface be modified or if another protocol be used, it will be necessary to rewrite management application 32 which would be undesirable. Portions of client API 34 would also have to be rewritten. In particular, having a management application 32 and API 34 that are depended upon COM protocol 36 presents difficulties when management application 32 is remote from computer system 10. In this scenario, it may be desirable to communicate over a network connection using any of a variety of protocols instead of always being required to use the COM protocol. In prior art computer system 10 management application 32 would have to be rewritten for each and every different protocol that is desired to be used.

[0010] Therefore, a technique is desired that would permit a management application to communicate both locally and remotely with a computer system using any of a variety of network protocols. It is desired to implement this technique with the least impact upon software developers and others responsible for the WBEM environment.

SUMMARY OF THE INVENTION

[0011] To achieve the foregoing, and in accordance with the purpose of the present invention, a transport neutral technique is disclosed that allows a management application to communicate with a computer system using any of a variety of network protocols. Advantageously, any management application software becomes independent of the transport mechanism used and need not be changed if the transport mechanism changes. Furthermore, such a transport neutral technique allows a client API to be used on any JAVA-enabled platform with any of a variety of transport mechanisms.

[0012] In one embodiment, a method is used for communication between a management application and a Common Information Model (CIM) object manager of a host computer. The method involves first creating a connection between the management application and a host computer. Next, a protocol indicator is passed from the management application to a client API. The protocol indicator identifies a protocol by which the management application desires to communicate with the host computer. A protocol-specific object is created having methods implemented using the protocol. Finally, the protocol-specific object is returned to the management application, thus the management application may communicate with the CIM object manager using the protocol desired.

[0013] In another embodiment, a computer system manages resources on a host computer. The computer system includes a management application that has program code for managing the host computer and a protocol indicator. Also included is a client application programming interface (client API) that has a factory class arranged to receive the protocol indicator from the management application and produce a protocol-specific object. Also within the client API is a first class having methods defined thereon implemented in a first protocol and a second class having methods defined thereon implemented in a second protocol. Thus, the protocol-specific object may be returned to the management application for use in managing the host computer.

BRIEF DESCRIPTION OF THE DRAWINGS

[0014] The invention, together with further advantages thereof, may best be understood by reference to the following description taken in conjunction with the accompanying drawings in which:

FIG. 1 illustrates a prior art computer system that has managed resources.

FIG. 2 illustrates a Web-Based Enterprise Management (WBEM) architecture suitable for implementing an embodiment of the invention.

FIGS. 3A and 3B illustrate an example graphical user interface for the CIM workshop of FIG. 2.

FIG. 4 illustrates an interface definition useful for implementing an embodiment of the invention.

FIG. 5 illustrates an implementation definition for the interface of FIG. 4.

FIG. 6 illustrates another implementation definition for the interface of FIG. 4.

FIG. 7 is a JAVA factory class definition.

FIG. 8 is a flowchart illustrating invocation of a method by a management application in a transport neutral manner.

FIGS. 9 and 10 illustrate a computer system suitable for implementing embodiments of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

[0015] FIG. 2 illustrates a Web-Based Enterprise Management (WBEM) architecture suitable for implementing an embodiment of the invention. Architecture 100 includes computer system 110 having resources to be managed, and application computer 150 where management applications are developed and run. Computer system 110 may be any suitable computer having resources needing to be managed such as CPU load, disk space, installed applications, etc. The hardware layer includes workstation 112 and CPU 114. Workstation 112 may be any suitable computer workstation such as the SPARC workstation from Sun Microsystems, Inc. CPU 114 may be any suitable processor such as an Intel processor. The software layer of computer system 110 includes operating system 116 (other software applications not shown for simplicity) which may be any suitable operating system such as the Solaris operating system from Sun Microsystems, Inc.

[0016] The object provider layer of computer system 110 includes operating system provider 118, operating system services 120 and provider application programming interface (API) 122. The provider layer communicates with operating system 116 via interface 117, for example, by using a JAVA Native Interface (JNI). Object providers act as intermediaries between CIM object manager 20 and one or more managed devices. When object manager 20 receives a request

from a management application 32 for data that is not available in CIM repository 130 it forwards the request to a provider. Object providers are installed on the same machine as object manager 20. Object manager 20 then uses provider API 122 to communicate with locally installed providers. Providers are classes that perform various functions in response to a request from object manager 20. For example, providers map information from a managed device to a CIM JAVA class and map information from a CIM JAVA class to a managed device format.

[0017] Provider API 122 is an API used by various provider programs to communicate information about managed objects to object manager 20. Operating system provider 118 is a collection of JAVA classes or native methods that represent the operating environment of computer system 110. Operating system services 120 also provide logging information from operating system 116 to object manager 20.

[0018] A variety of other providers may also be present. For example, an SNMP provider includes JAVA classes that map CIM data to SNMP data. Also, a CPU-specific provider may be used to transport resource information directly between CPU 114 and provider API 122, thus bypassing operating system 116.

[0019] Providers may be categorized into three types according to the requests they service. An instance type supplies dynamic instances of a given class and supports the retrieval, enumeration, modification and deletion operations. A property type supplies dynamic property values, for example, disk space. A method type supplies methods of one or more classes. A single provider can support both methods and instances. Most providers are "pull" providers which mean they maintain their own data, generating it dynamical if necessary. Pull providers have minimal interaction with object manager 20 and CIM repository 130. The data managed by a pull provider typically changes frequently, requiring the provider to either generate the data dynamically or retrieve it from a local cache whenever an application issues a request. A single provider can act simultaneously as a class instance and method provider by proper registration and implementation of all relevant methods.

[0020] The management layer of computer system 110 includes CIM object manager 20, CIM repository 130 and web server software 140. Object manager 20 may be any suitable WBEM compliant manager. Object manager 20 manages CIM objects that are represented internally as JAVA classes. Client computers running management applications (such as application computer 150) connect to object manager 20 for resource information about computer system 110. When a WBEM client connects to object manager 20 it receives a reference to that object manager. The client can then perform WBEM operations using this reference.

[0021] When management application 32 uses client API 156 to request or update information about a

managed object, object manager 20 contacts either the appropriate provider for that object or a suitable persistent storage mechanism such as repository 130. In one embodiment, classes that are handled by a provider have a "provider" qualifier that identifies the provider to contact for the class. When object manager 20 receives a request for a class that has a "provider" qualifier, it routes the request to the specified provider. If no provider is specified it routes the request to repository 130 using JAVA Naming and Directory Interface (JNDI) 132.

[0022] Object manager 20 also performs various start-up functions: starting and registering the RMI server; registering the XML server; setting up a connection to repository 130; and waiting for incoming requests. Object manager 20 also performs other normal operations: performing security checks such as authentication and authorization; performing syntactical and semantic checking of CIM data operations; routing requests to providers or persistent storage; and delivering data from providers or from persistent storage to client management applications.

[0023] CIM repository 130 is a central storage area for CIM class and instance definitions. In one embodiment of the invention, repository 130 uses Sun Directory Services that support a LightWeight Directory Access Protocol (LDAP) and communicates with object manager 20 using a JNDI protocol. In the present embodiment, WBEM classes are stored in repository 130 as JAVA serialized objects; LDAP attributes and values may also be stored.

[0024] Web server software 140 may be any suitable WBEM XML-compliant web server such as the Sun Web Server available from Sun Microsystems, Inc. JAVA servlet 142 converts XML data to the client API format. For example, if management application 132 contains XML data, client API client 156 encodes the data as XML messages and transports the encoded messages to web server 140 that is running JAVA servlet 142. Web server 140 listens for XML messages on a standard port and passes control to servlet 142 when detected. Servlet 142 then decodes the XML messages it receives. Servlet 142 then converts the XML data to the client API format and transmits the information back to client API 156 in RMI format. Alternatively, should object manager 20 support the HTTP format, client API 156 may communicate directly to object manager 20 without the need for web server 140.

[0025] The application layer of WBEM architecture 100 includes management application 32, CIM workshop 152, MOF compiler 154 and client application programming interface (API) 156. In this embodiment of the invention, these elements of the application layer are shown running on application computer 150 (other hardware and software not shown for simplicity). Alternatively, application computer 150 and computer system 110 may be the same computer or the elements of the application layer may reside on a variety of computers and not exclusively on application computer 150.

[0026] A software developer 30 uses any suitable software tool to develop a management application 32 for processing and displaying data from managed objects of computer system 110. Management application 32 uses client API 156 to request information about managed objects from object manager 20. In this fashion, analysis of the resources of computer system 110 can be presented to a system administrator 40 for proper action.

[0027] Client API 156 and provider APIs represent and manipulate CIM objects. These APIs represent CIM objects as JAVA classes. An object is a computer representation or model of a managed resource of computer system 110 such as a printer, disk drive or CPU. A developer uses the CIM specification to describe managed objects and to retrieve information about managed objects in computer system 110. One advantage of modeling managed resources using CIM is that those objects can be shared across any system that is CIM compliant.

[0028] Management application 32 may be any of a wide variety of software applications written to analyze and manage the resources of computer system 110. By way of example, management application 32 manages system aspects such as disk information (space available, partitions, etc.), CPU load, event processing, date, time, time zone, memory available, ports, etc.

[0029] Application 32 may also manage specific devices of the computer system such as disks, tape drives, modems, other I/O devices, NICs, and network aspects of the system such as TCP/IP, Netbeui, Novell, etc. Further, management application 32 manages the software applications running on computer system 110 by determining what is currently running on the system, what is currently installed, the state of installation, which applications can be terminated, performing application metering, managing application life cycle, process management, user management, etc.

[0030] In one embodiment of the invention, developer 30 uses a CIM workshop 152 written in JAVA for viewing, changing, adding and deleting CIM classes and instances. CIM workshop 152 provides a graphical user interface for the developer. For example, developer 30 may view and select namespaces, may add namespaces, add properties, qualifiers and methods to new classes, view and create instances, and view and modify instance values. Developer 30 may also use CIM workshop 152 to browse a class inheritance tree and change the root of an object tree for a namespace.

[0031] MOF compiler 154 parses files created in the Managed Object Format (MOF), converts files to JAVA class and stores the extracted classes and instances in repository 130. The MOF language is a syntax for defining CIM classes and instances and is described in the CIM specification. Although classes and instances can also be added through client API 156 using JAVA, MOF compiler 154 eliminates the need to write such code. Compiler 154 provides developers and administrators

with a simple and fast technique for modifying repository 130.

[0032] Client API 156 is an application programming interface used by management application 32 to communicate with object manager 20 using Remote Method Invocation (RMI) protocol 158 or XML over an HTTP protocol 160. Other suitable protocols may also be used. Client API 156 may communicate directly with object manager 20 using RMI or may communicate using the XML/HTTP protocol using web server 140. Alternatively, client API 156 can communicate using the XML/HTTP protocol 160 directly should object manager 20 support the HTTP protocol. In one embodiment, client API 156 is a public API that JAVA applications use to request operations from object manager 20. Client API 156 is used by management application 32 to transfer data to and from object manager 20. Client API 156 includes a variety of classes, instances and methods useful for communicating with object manager 20 using any suitable transport mechanism; these are discussed below in FIGS. 4-8.

[0033] Connections 170a-170d are any suitable local or network connection between computer system 110 and application computer 150. By way of example, these connections occur over an internet, an intranet, an extranet, within a workgroup, or other.

[0034] FIGS. 3A and 3B illustrate an example graphical user interface for CIM workshop 152. Preferably, a login to the workshop prompts for a host name, namespace, user name and password. By default, workshop 152 connects to the object manager on the local host in the default namespace. FIG. 3A illustrates CIM classes that represent objects in the selected namespace on the selected host. Listed in panel 210 are the objects of the selected namespace. On the right-hand panel are shown the properties 212 for the selected object (in this case the object "Solaris Package") and methods 214 (not shown). FIG. 3B shows all instances of a selected object. Instances are shown in the left-hand panel, and in this example instance 252 is shown. The right-hand panel shows all properties 254 associated with the selected instance and its associated methods 256 (not shown).

[0035] FIG. 4 illustrates an interface definition 300 of Client API 156 useful for implementing an embodiment of the invention. Interface 300 lists various methods that may be called by management application 32 in the course of managing resources of computer system 110. Advantageously, this interface may be implemented using a variety of classes having protocol-specific methods thus allowing transport neutral management applications to be written. Interface 300 includes an interface name 302 which in this example is "CIM Client API." Included are a variety of methods defined for the interface. Each method has a method name 304, a return value 306 and parameters 308. By way of example, shown is one method "Create Namespace" 310 having a return value of "void" and accept-

ing parameters version and namespace. A large number of other methods may be defined for interface 300.

[0036] By way of example, these methods include the following. The Create Namespace method creates a CIM namespace, a directory containing classes and instances. When a management application connects to object manager 20 it specifies a namespace. All subsequent operations occur within that namespace on the object manager host. The method Delete Namespace deletes the specified namespace on the specified host. The method Delete Class deletes the specified class. The method Delete Instance deletes the specified instance. The method Delete Qualifier Type deletes the specified qualifier type. The method Enumerate Class retrieves the specified class from object manager 20. The method Enumerate Namespace gets a list of namespaces. The method Enumerate Instances gets a list of instances for the specified class. The method Enumerate Qualifier Types get a list of qualifier types for the specified class. The method Get Class gets the CIM class for the specified CIM object path. The Get Instance method gets the CIM instance for the specified CIM object path. The method Invoke Method executes the specified method on the specified object.

[0037] The method Get Qualifier Type gets the qualifier type for the specified CIM object path. The method Set Class invokes object manager 20 to modify the specified CIM class to the specified namespace. The method Set Instance invokes the CIM object manager to add or update the specified CIM instance to the specified namespace. The Set Qualifier Type method invokes the CIM object manager to add the specified qualifier type to the specified namespace. The method Create Class creates a class definition of the specified type. The method Create Instance creates a new instance of the specified type. The method Create Qualifier Type creates a new qualifier type. The method Get Property gets a property of the specified instance. The method Set Property sets a property of the specified instance. The method Close closes the client connection to object manager 20. This method frees resources used for the client session. Other methods may also be included within interface 300.

[0038] Once interface 300 has been defined it is possible to then code protocol-specific methods to implement each of the methods defined in interface 300. In this fashion, any number of protocol-specific classes are provided each having an implementation for a specific protocol such as RMI or XML. Though the use of these protocol-specific classes, management application 32 is able to communicate with object manager 20 over any suitable protocol in a transparent fashion.

[0039] FIG. 5 illustrates an implementation definition 400 for the interface of FIG. 4. Specifically, implementation 400 implements class "CIM Client API" using methods specific to the RMI protocol. Such protocol-specific methods allow management application 32 to

communicate via client API 156 to object manager 20 using the RMI protocol. Implementation 400 includes a class name 402 "CIM client RMI." Also included is constructor definition code 404 that constructs an instance of class 402 that is specific to the RMI protocol. Use of a constructor definition to create an instance is well known to those of skill in the art.

[0040] Also included in implementation 400 are the specific implementations of the methods defined upon interface 300. For each method implemented there is a method name 406, a return value 408, parameters 410 and implementation code 412. Implementation code 412 is preferably JAVA code that implements the particular method using any constructs necessary that are specific to the RMI protocol. Those of skill in the art will appreciate how to implement JAVA code for a particular purpose that must adhere to a specific protocol.

[0041] Preferable, all of the methods defined upon interface 300 are implemented in implementation 400. Shown by way of example is the method Create Name Space 416 which has a return value of "void" and accepts the parameters version and namespace. Not shown for simplicity is the actual RMI-specific JAVA code that implements the method Create Namespace. The other methods defined in interface 300 are also listed in implementation 400 along with their RMI-specific code.

[0042] FIG. 6 illustrates an implementation definition 500 for the interface of FIG. 4. Specifically, implementation 500 implements class "CIM Client API" using methods specific to the XML protocol. Such protocol-specific methods allow management application 32 to communicate via client API 156 to object manager 20 using the XML protocol. Implementation 500 includes a class name 502 "CIM client XML." Also included is constructor definition code 504 that constructs an instance of class 502 that is specific to the XML protocol. Use of a constructor definition to create an instance is well known to those of skill in the art.

[0043] Also included in implementation 500 are the specific implementations of the methods defined upon interface 300. For each method implemented there is a method name 506, a return value 508, parameters 510 and implementation code 512. Implementation code 512 is preferably JAVA code that implements the particular method using any constructs necessary that are specific to the XML protocol. Those of skill in the art will appreciate how to implement JAVA code for a particular purpose that must adhere to a specific protocol.

[0044] Preferable, all of the methods defined upon interface 300 are implemented in implementation 500. Shown by way of example is the method Create Name Space 516 which has a return value of "void" and accepts the parameters version and namespace. Not shown for simplicity is the actual XML-specific JAVA code that implements the method Create Namespace. The other methods defined in interface 300 are also listed in implementation 500 along with their XML-spe-

cific code.

[0045] FIG. 7 is a JAVA factory class 600. Factory 600 is used for determining which protocol is desired by management application 32 and directing the creation of a protocol-specific object to be returned to the management application.

[0046] Factory 600 includes a class name 602 "CIM Client Factory" and any number of defined methods. For each method there is a method name 604, a return value 606, parameters 608 and an implementation 610. In particular, the method Get Client API accepts the parameters protocol, namespace and version, and returns an instance of interface 300 which is a protocol-specific instance of either implementation 400 or implementation 500. Of course, other protocol-specific objects may be returned if other implementations are defined. The implementation code 610 for method 612 may be any suitable JAVA code that checks the protocol parameter to see which protocol is desired and then directs either implementation 400 or 500 to construct a new instance of itself. By way of example, a series of case statements may be used. Other methods may also be defined and implemented within factory 600.

MANAGEMENT APPLICATION EXECUTION

[0047] FIG. 8 is a flowchart illustrating invocation of a method by a management application. Once management application 32 has been created by developer 30 and the class and methods of FIGS. 4-7 have been defined, management application 32 may execute and request resources of computer system 110 using any desired protocol. FIG. 8 illustrates a single method call according to one embodiment of the invention.

[0048] In step 702 management application 32 creates a connection from application computer 150 to computer system 110. Preferably, application 32 invokes a method within Client API 156 which creates an instance of application 32 within object manager 20. Application 32 passes to the method a host name, a namespace, a user name, a password, and the protocol by which it is desired to communicate with host computer system 110. Any suitable protocol may be identified such as RMI, XML/HTTP, DCOM or any other suitable network communication protocol.

[0049] In step 706 factory 600 of Client API 156 checks the protocol desired by application 32 using its method Get Client API. This method returns a protocol-specific object which is an instance of either the class defined in implementation 400 or the class defined in implementation 500. For example, in step 710 if the protocol parameter is RMI, then in step 714 the constructor definition 404 of implementation 400 executes and results in an RMI-specific object having RMI-specific methods being returned to application 32. On the other hand, if the protocol parameter is XML, then in step 722 the constructor definition 504 of implementation 500 executes and produces an XML-specific object which is

returned to application 32. As shown in steps 726 and 730, other protocols are also supported. In step 738 application 32 invokes a desired management method upon the protocol-specific object recently returned. Because the methods of this object are specific to the protocol desired by application 32, communication between Client API 156 and object manager 20 occurs using the desired protocol in a fashion transparent to application 32.

[0050] Once object manager 20 has processed the method (which may be a request for an object, request for information, or a command to computer system 110), then in step 742 the result is returned from object manager 20 to application 32 via Client API 156 using the desired protocol. In this fashion, a transport neutral technique has been described that allows a management application to be written independent of the protocol by which it is desired to communicate with a target computer system.

COMPUTER SYSTEM EMBODIMENT

[0051] FIGS. 9 and 10 illustrate a computer system 900 suitable for implementing embodiments of the present invention. FIG. 9 shows one possible physical form of the computer system. Of course, the computer system may have many physical forms ranging from an integrated circuit, a printed circuit board and a small handheld device up to a huge super computer. Computer system 900 includes a monitor 902, a display 904, a housing 906, a disk drive 908, a keyboard 910 and a mouse 912. Disk 914 is a computer-readable medium used to transfer data to and from computer system 900.

[0052] FIG. 10 is an example of a block diagram for computer system 900. Attached to system bus 920 are a wide variety of subsystems. Processor(s) 922 (also referred to as central processing units, or CPUs) are coupled to storage devices including memory 924. Memory 924 includes random access memory (RAM) and read-only memory (ROM). As is well known in the art, ROM acts to transfer data and instructions unidirectionally to the CPU and RAM is used typically to transfer data and instructions in a bi-directional manner. Both of these types of memories may include any suitable of the computer-readable media described below. A fixed disk 926 is also coupled bi-directionally to CPU 922; it provides additional data storage capacity and may also include any of the computer-readable media described below. Fixed disk 926 may be used to store programs, data and the like and is typically a secondary storage medium (such as a hard disk) that is slower than primary storage. It will be appreciated that the information retained within fixed disk 926, may, in appropriate cases, be incorporated in standard fashion as virtual memory in memory 924. Removable disk 914 may take the form of any of the computer-readable media described below.

[0053] CPU 922 is also coupled to a variety of

input/output devices such as display 904, keyboard 910, mouse 912 and speakers 930. In general, an input/output device may be any of: video displays, track balls, mice, keyboards, microphones, touch-sensitive displays, transducer card readers, magnetic or paper tape readers, tablets, styluses, voice or handwriting recognizers, biometrics readers, or other computers. CPU 922 optionally may be coupled to another computer or telecommunications network using network interface 940. With such a network interface, it is contemplated that the CPU might receive information from the network, or might output information to the network in the course of performing the above-described method steps. Furthermore, method embodiments of the present invention may execute solely upon CPU 922 or may execute over a network such as the Internet in conjunction with a remote CPU that shares a portion of the processing.

[0054] In addition, embodiments of the present invention further relate to computer storage products with a computer-readable medium that have computer code thereon for performing various computer-implemented operations. The media and computer code may be those specially designed and constructed for the purposes of the present invention, or they may be of the kind well known and available to those having skill in the computer software arts. Examples of computer-readable media include, but are not limited to: magnetic media such as hard disks, floppy disks, and magnetic tape; optical media such as CD-ROMs and holographic devices; magneto-optical media such as floptical disks; and hardware devices that are specially configured to store and execute program code, such as application-specific integrated circuits (ASICs), programmable logic devices (PLDs) and ROM and RAM devices. Examples of computer code include machine code, such as produced by a compiler, and files containing higher level code that are executed by a computer using an interpreter.

[0055] Although the foregoing invention has been described in some detail for purposes of clarity of understanding, it will be apparent that certain changes and modifications may be practiced within the scope of the appended claims. For instance, the application computer and the computer system to be managed may be the same computer, or may be separated by a great distance. The use of a web server may not be required should the CIM object manager support the HTTP format. Other types of classes and methods may be used while not departing from the spirit of the invention. Therefore, the described embodiments should be taken as illustrative and not restrictive, and the invention should not be limited to the details given herein but should be defined by the following claims and their full scope of equivalents.

Claims

1. A method for communication between a management application and a Common Information Model (CIM) object manager of a host computer, said method comprising:
 - creating a connection between said management application and said host computer;
 - passing a protocol indicator from said management application to a client API, said protocol indicator identifying a protocol by which said management application desires to communicate with said host computer;
 - creating a protocol-specific object having methods implemented using said protocol; and
 - returning said protocol-specific object to said management application, whereby said management application may communicate with said CIM object manager using said protocol.
2. The method of claim 1 further comprising:
 - invoking a method defined upon said protocol-specific object;
 - transmitting said method using said protocol over said connection to said CIM object manager of said host computer; and
 - returning a result to said management application over said connection using said protocol.
3. The method of claim 1 wherein said protocol is RMI, XML or DCOM.
4. The method of claim 1 wherein said management application is resident on said host computer.
5. The method of claim 1 wherein said management application is resident on a separate application computer.
6. The method of claim 1 wherein said creating a protocol-specific object includes calling a JAVA factory class.
7. A computer system for managing resources of a host computer, said system comprising:
 - a management application including a protocol indicator and program code for managing said host computer; and
 - a client application programming interface (cli-

ent API) including

a factory class arranged to receive said protocol indicator from said management application and produce a protocol-specific object,

a first class having methods defined thereon implemented in a first protocol, and

a second class having methods defined thereon implemented in a second protocol, whereby said protocol-specific object may be returned to said management application for use in managing said host computer.

8. The system of claim 7 wherein said host computer includes a Common Information Model (CIM) object manager arranged to receive a method call from said management application using the protocol identified by said protocol indicator.
9. The system of claim 7 wherein said computer system and said host computer are the same computer.
10. The system of claim 7 wherein said computer system and said host computer are connected over a network connection implemented in the protocol identified by said protocol indicator.
11. The system of claim 7 wherein the protocol identified by said protocol indicator is RMI, XML or DCOM.
12. A computer-readable medium comprising computer code for communication between a management application and a Common Information Model (CIM) object manager of a host computer, said computer code of said computer-readable medium effecting the following:
 - creating a connection between said management application and said host computer;
 - passing a protocol indicator from said management application to a client API, said protocol indicator identifying a protocol by which said management application desires to communicate with said host computer;
 - creating a protocol-specific object having methods implemented using said protocol; and
 - returning said protocol-specific object to said management application, whereby said management application may communicate with

said CIM object manager using said protocol.

13. The computer-readable medium of claim 12 further comprising computer code for effecting the following:

invoking a method defined upon said protocol-specific object;

transmitting said method using said protocol over said connection to said CIM object manager of said host computer; and

receiving a result destined for said management application over said connection using said protocol.

14. The computer-readable medium of claim 12 wherein said protocol is RMI, XML or DCOM.
15. The computer-readable medium of claim 12 wherein said creating a protocol-specific object includes
 - calling a JAVA factory class.

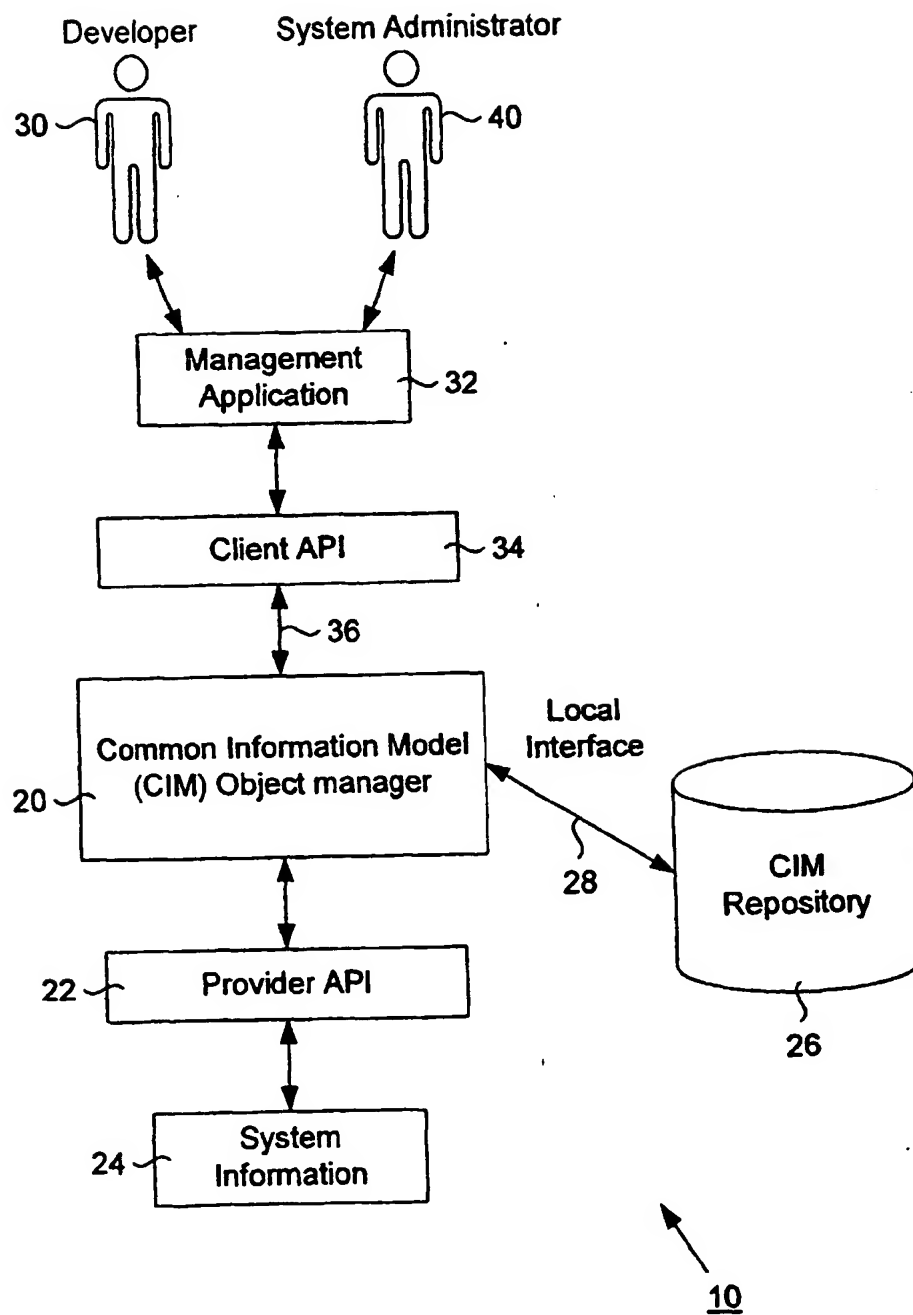


FIG. 1
(Prior Art)

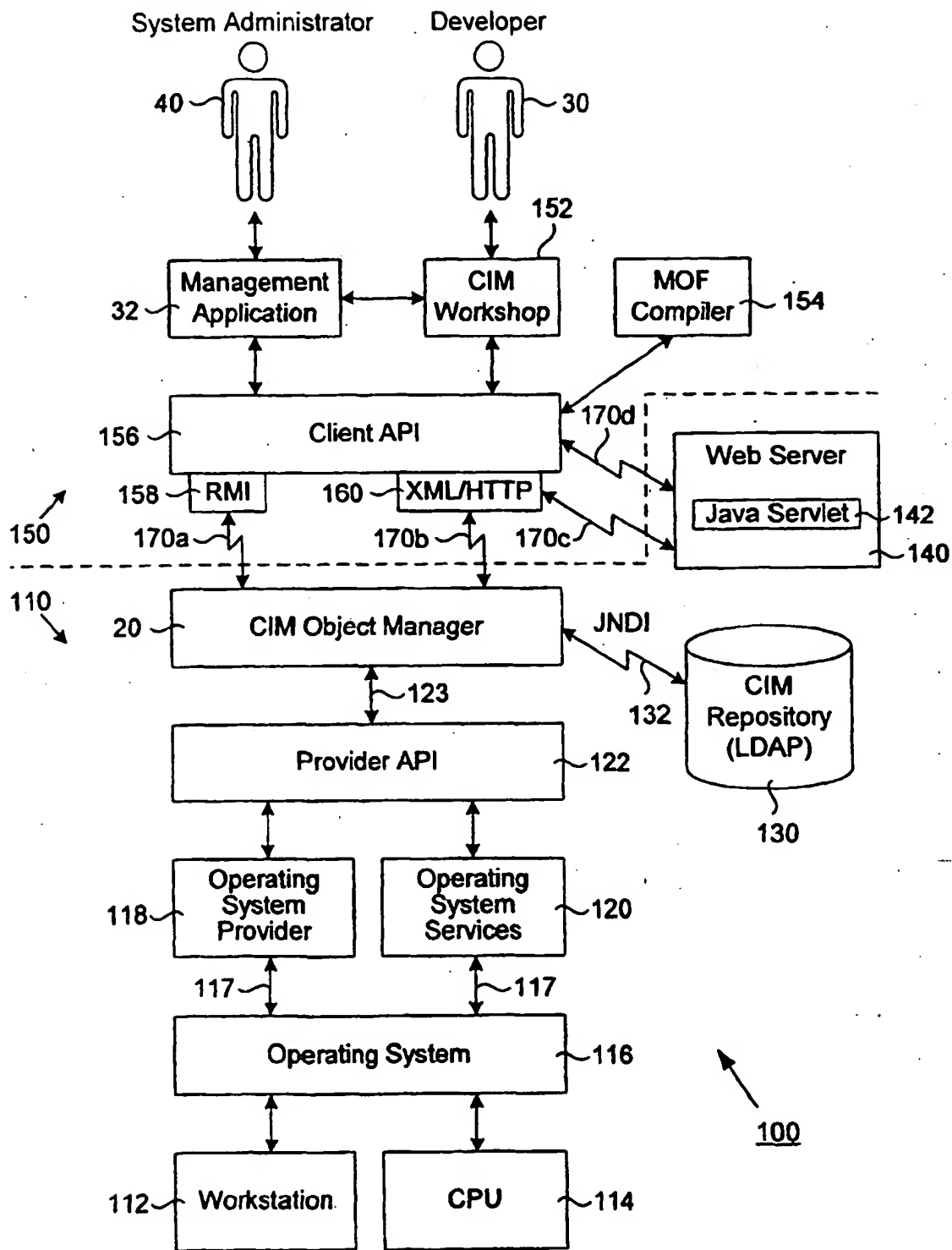


FIG. 2

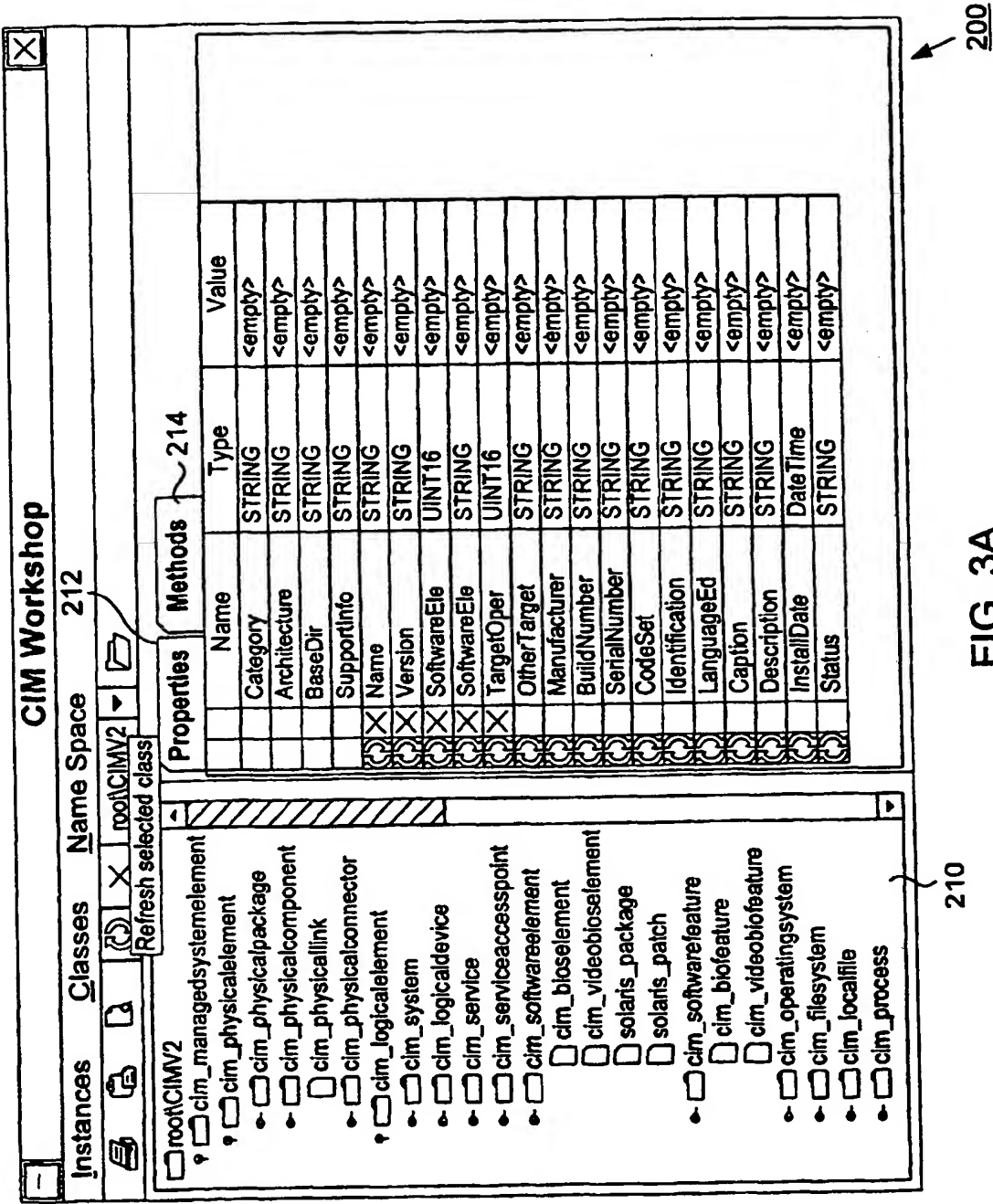


FIG. 3A

254

Instances

252

CSCreationClassName

Properties Methods 256

	Name	Type	Value
<input checked="" type="checkbox"/>	CSCreationClassName	STRING	Solaris_ComputerSystem
<input checked="" type="checkbox"/>	CSName	STRING	zoomass
<input checked="" type="checkbox"/>	CreationClassName	STRING	Solaris_OperatingSystem
<input checked="" type="checkbox"/>	Name	STRING	SunOS
<input checked="" type="checkbox"/>	OSType	UINT16	29
<input checked="" type="checkbox"/>	OtherTypeDescription	STRING	<empty>
<input checked="" type="checkbox"/>	Version	STRING	<empty>
<input checked="" type="checkbox"/>	LastBootUp Time	Date Time	<empty>
<input checked="" type="checkbox"/>	LocalDate Time	Date Time	<empty>
<input checked="" type="checkbox"/>	CurrentTimeZone	SINT16	<empty>
<input checked="" type="checkbox"/>	NumberOfLicensedUsers	UINT32	<empty>
<input checked="" type="checkbox"/>	NumberOfUsers	UINT32	<empty>
<input checked="" type="checkbox"/>	NumberOfProcesses	UINT32	<empty>
<input checked="" type="checkbox"/>	MaxNumberOfProcesses	UINT32	<empty>
<input checked="" type="checkbox"/>	TotalSwapSpaceSize	UINT64	<empty>
<input checked="" type="checkbox"/>	TotalVirtualMemorySize	UINT64	<empty>
<input checked="" type="checkbox"/>	FreeVirtualMemory	UINT64	<empty>
<input checked="" type="checkbox"/>	FreePhysicalMemory	UINT64	<empty>
<input checked="" type="checkbox"/>	SizeStoredInPagingFiles	UINT64	<empty>

FIG. 3B

250

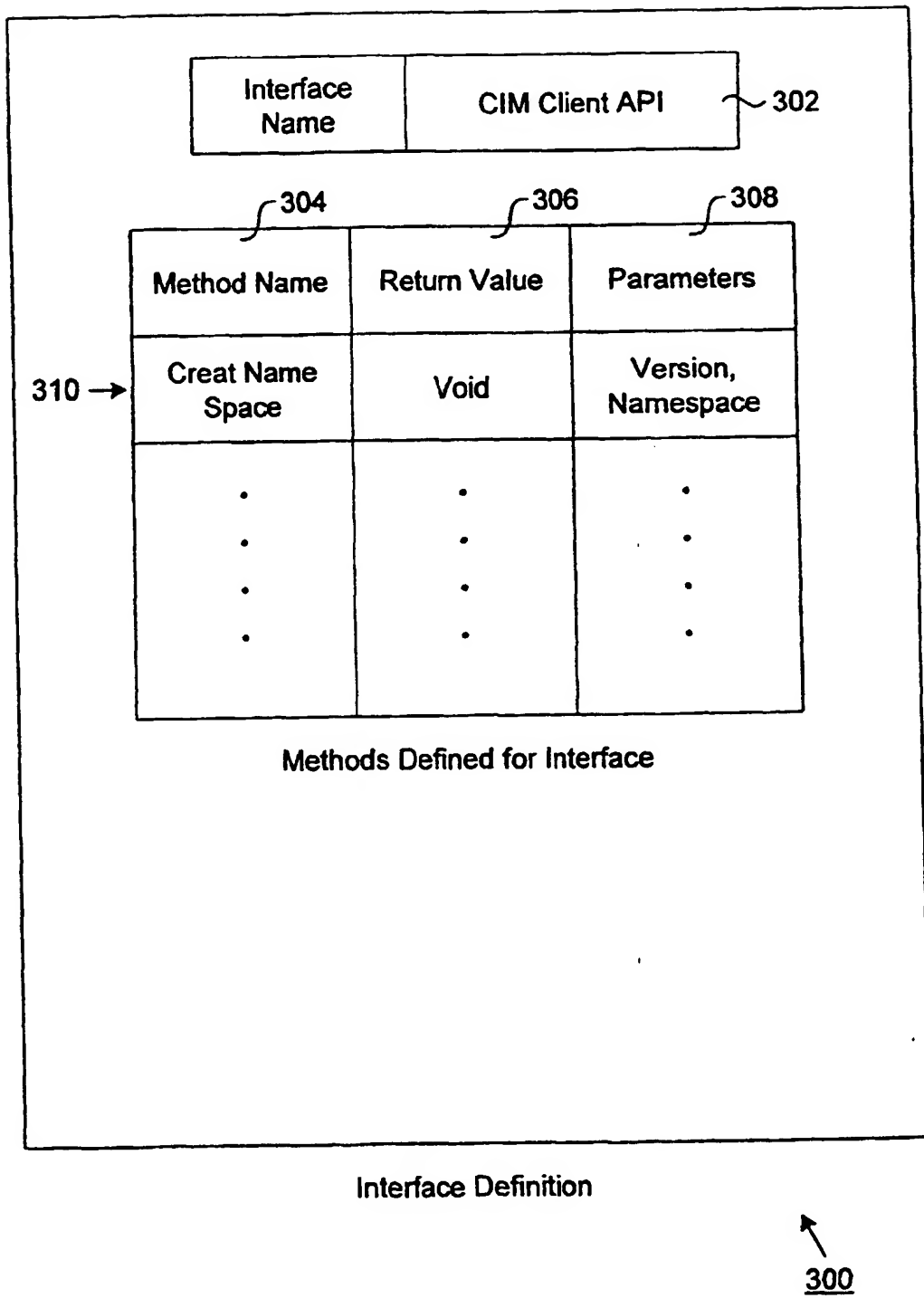
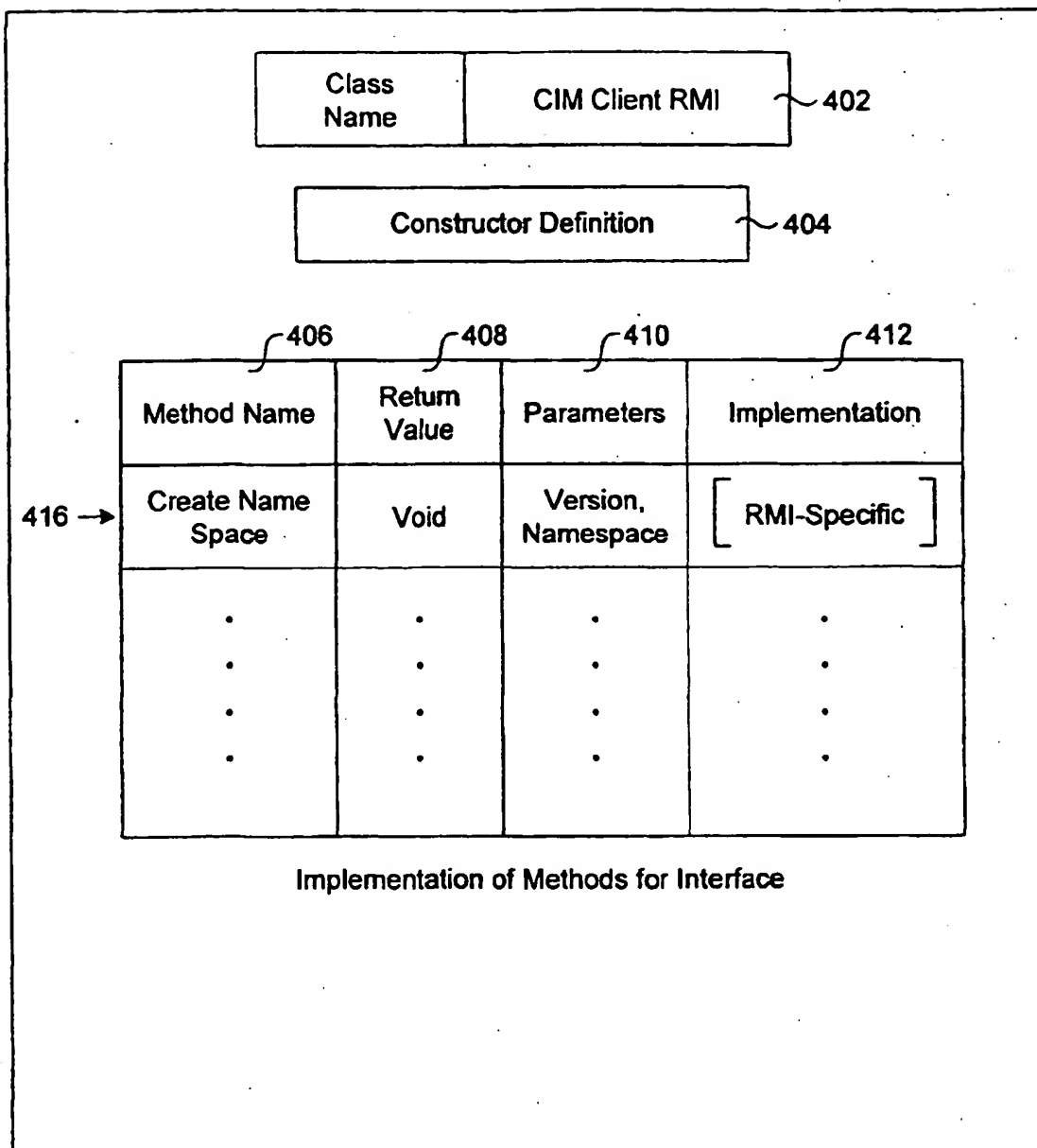


FIG. 4



Implementation Definition for Interface

400

FIG. 5

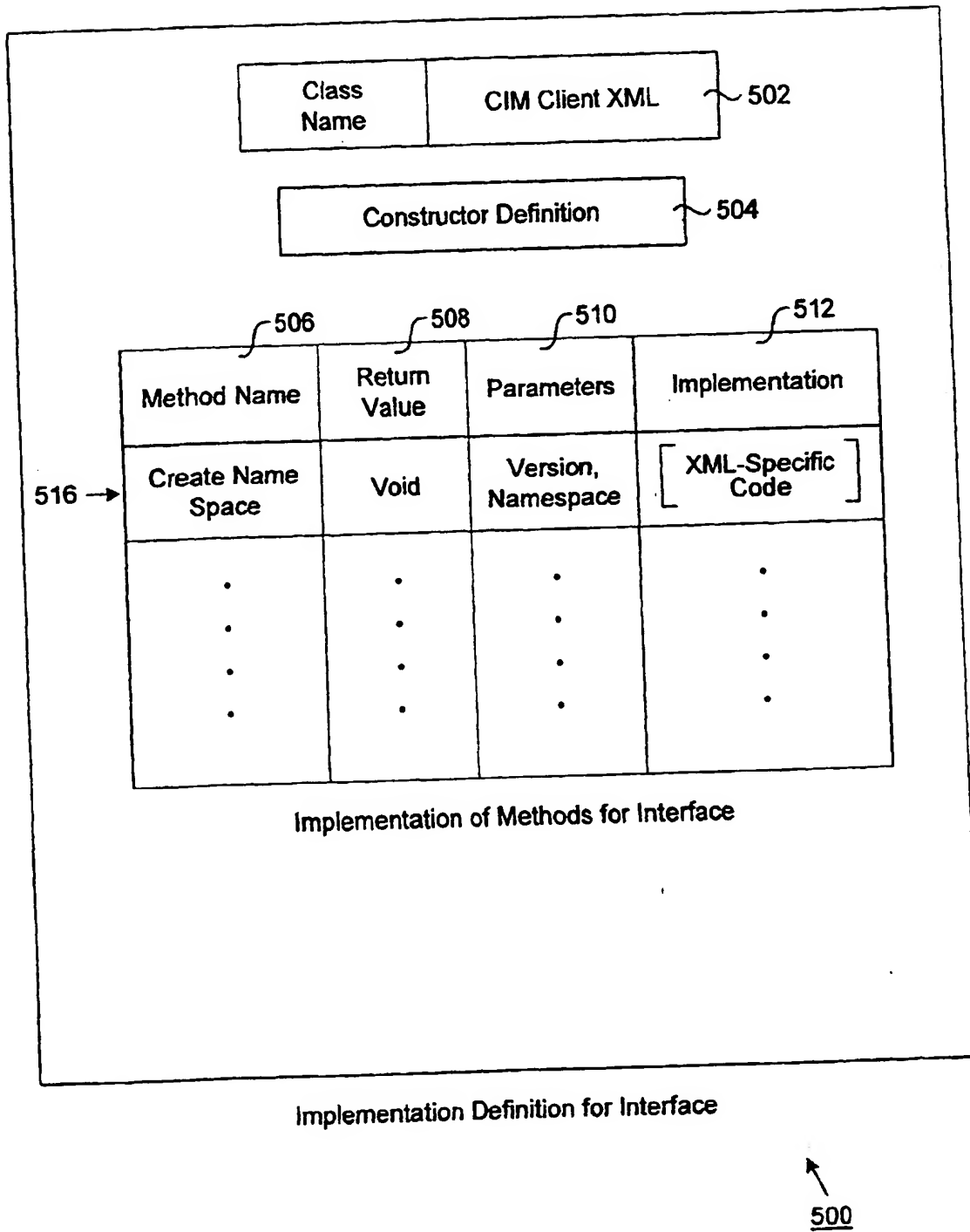


FIG. 6

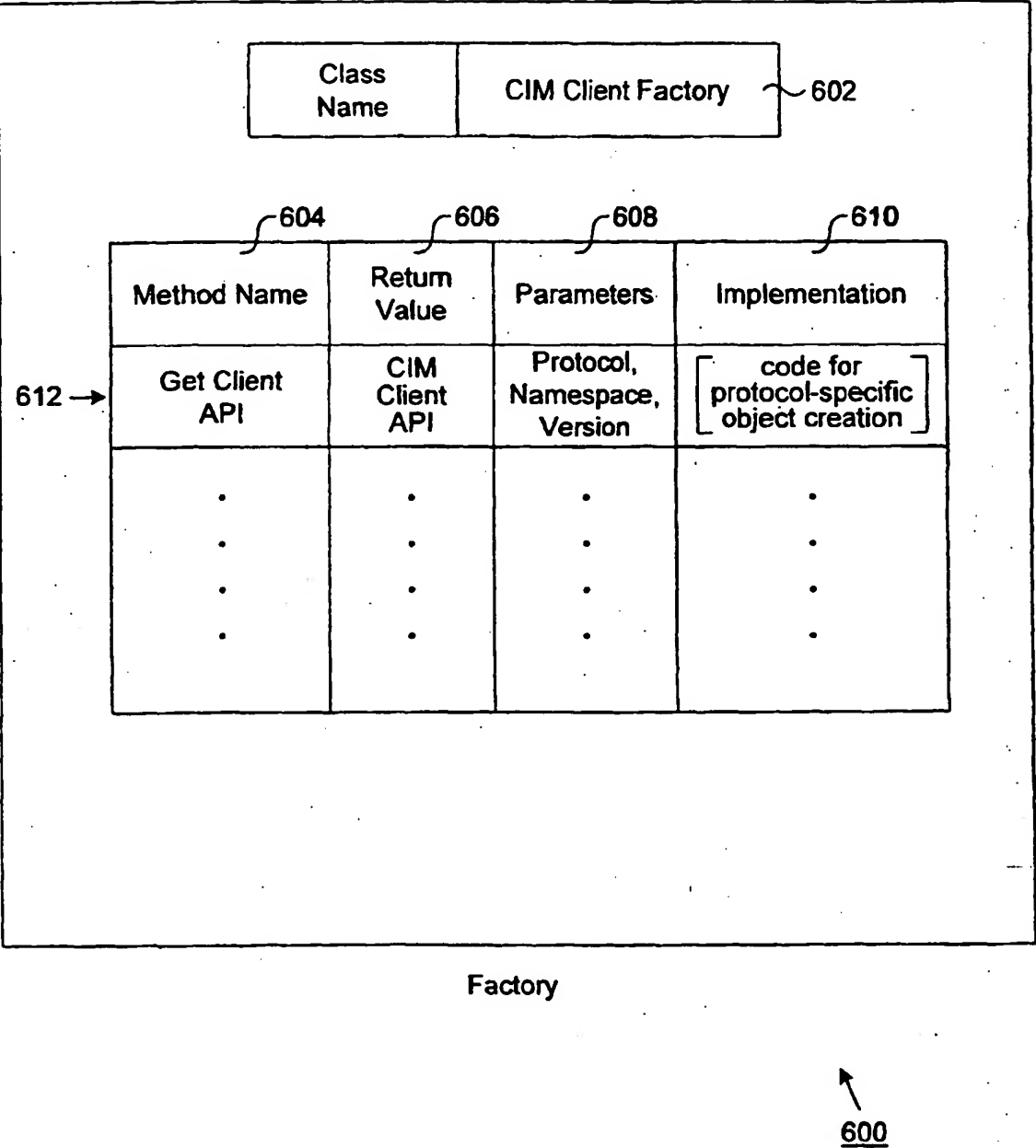


FIG. 7

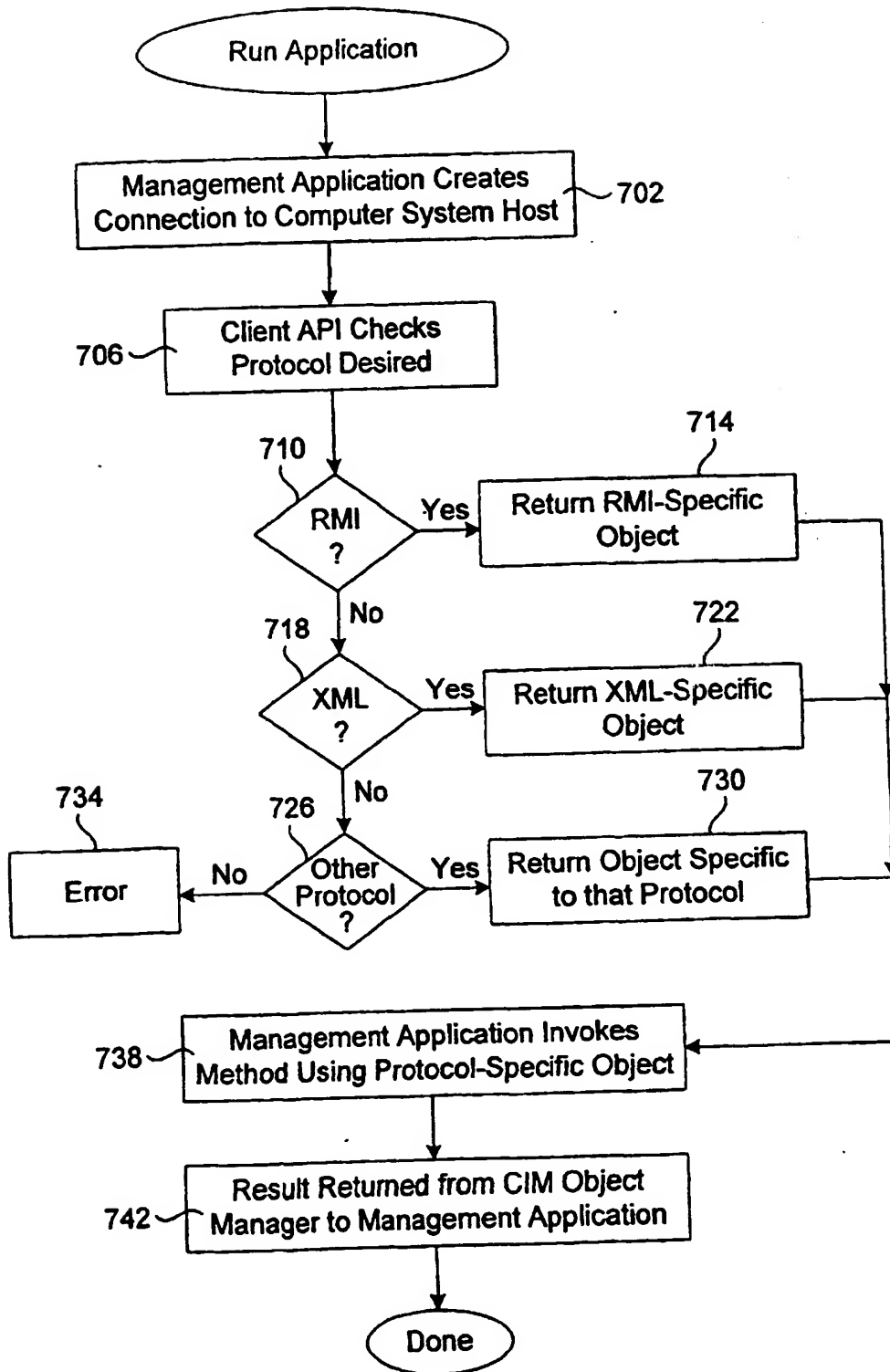


FIG. 8

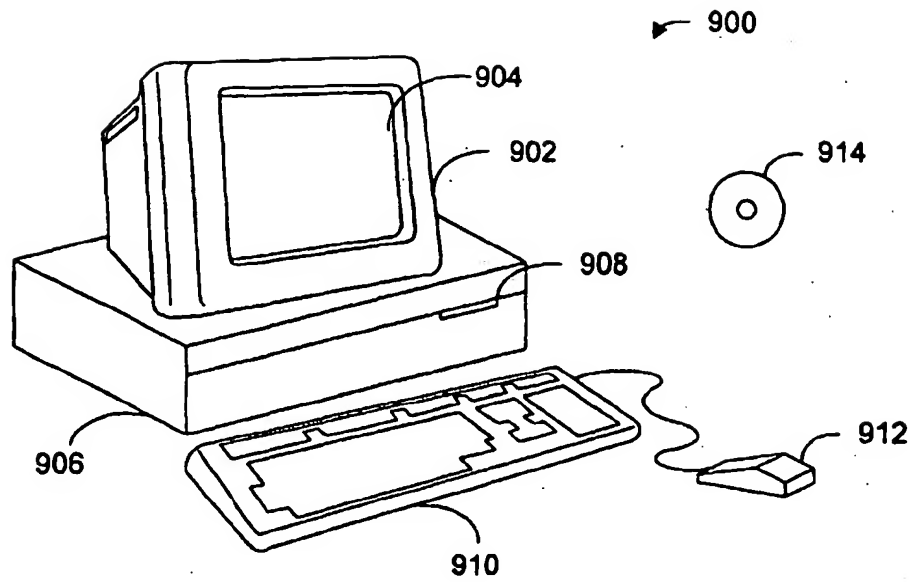


FIG. 9

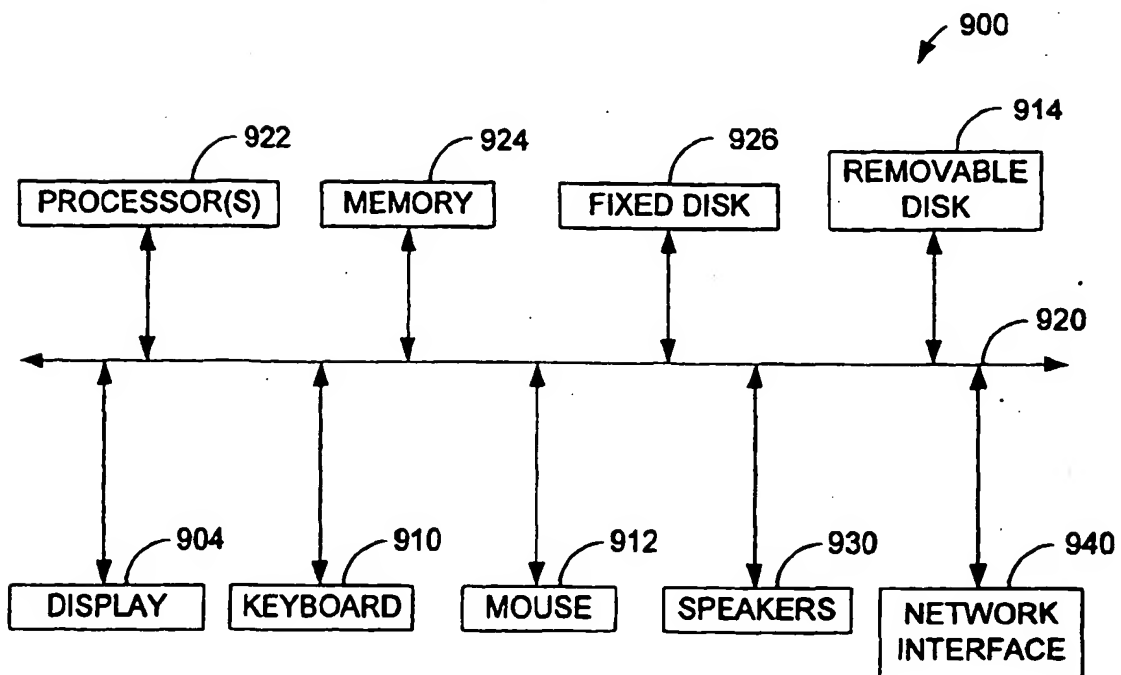


FIG. 10

THIS PAGE BLANK (USPTO)